Physically Based Rendering (4th Ed): Chapter 4 Summary

Radiometry, Spectra, and Color: The Physics of Light

Gemini (For a Developer Audience)

November 29, 2025

Introduction: Light is Just Data

In Chapter 2, we learned that rendering is just "averaging random samples." But what exactly are we sampling? We are sampling **Light**.

Chapter 4 is about **Radiometry**, which is the system of units we use to measure light. If you are a programmer, think of this as defining your **Data Types**. You wouldn't mix up an 'int' and a 'string'. Similarly, you shouldn't mix up "Flux" and "Radiance."

This chapter answers: "What number do I put in my 'Spectrum' class?"

1 Radiometry: The Data Types

There are four main units you need to know. They all measure "light," but in different contexts.

1.1 1. Radiant Energy (Q)

Unit: Joules (J)

This is raw energy. It's the total amount of energy a photon carries.

- **Code Analogy**: A battery's total capacity.
- **Usage**: Rarely used directly in rendering because we care about light *over time* (frames per second), not total energy over a century.

1.2 2. Radiant Flux (Φ)

Unit: Watts (W) or Joules/second

This is energy flowing through space **per second**.

- **Code Analogy**: Bandwidth (Megabytes per second).
- **Usage**: The total power of a light source. A "100 Watt lightbulb" is emitting 100 Joules of energy every second.
- **Math**: $\Phi = \frac{dQ}{dt}$ (Change in energy over change in time).

1.3 3. Irradiance (E)

Unit: Watts per square meter (W/m^2)

This is Flux arriving at a **Surface**.

- **Code Analogy**: Rain falling on the ground.
- **Usage**: How much light is hitting this specific polygon?
- **Math**: $E = \frac{d\Phi}{dA}$ (Flux per unit Area).

If you have a 100W lightbulb (Φ) and you spread its light over a huge wall (A), the Irradiance (E) is low. The wall is dim. If you focus it on a tiny dot, E is huge. The dot burns.

1.4 4. Radiance (L)

Unit: Watts per steradian per square meter ($W/(sr \cdot m^2)$)

This is the big one. **Radiance is what you see.**

- **Code Analogy**: A pixel on your monitor.
- **Usage**: Light traveling along a **specific ray**.
- **Math**: $L = \frac{d^2\Phi}{d\omega\, dA^\perp}$

Why is Radiance Special?

Radiance has a magical property: **It is constant along a ray.**
If you look at a wall from 1 meter away, or 100 meters away, the brightness (Radiance) of the point you are looking at is the same.

- **Flux** falls off with distance ($1/r^2$).
- **Irradiance** falls off with distance.
- **Radiance** stays constant.

This is why Ray Tracing works. We trace a ray from the camera to the wall. We calculate the Radiance leaving the wall. We assign that value directly to the pixel. We don't need to worry about how far away the camera is.

2 Working with Integrals

You will see this scary equation everywhere in PBRT:

$$L_o(p, \omega_o) = \int_\Omega f(p, \omega_o, \omega_i) L_i(p, \omega_i) |\cos \theta_i| d\omega_i \tag{1}$$

Don't panic. Let's parse it like a function definition.

- $L_o(p, \omega_o)$: **Output**. Light leaving point p in direction ω_o (towards camera).
- $\int_\Omega$: **For Loop**. Loop over every possible incoming direction (the hemisphere Ω).
- $L_i(p, \omega_i)$: **Input**. Light arriving at point p from direction ω_i.
- $f(p, \omega_o, \omega_i)$: **Material**. The BRDF (Bidirectional Reflectance Distribution Function). It says "If light comes from ω_i, how much reflects to ω_o?"
- $|\cos \theta_i|$: **Geometry Factor**. Light hitting a surface at a steep angle spreads out more (Lambert's Law).

The Programmer's Translation:

```
Color Lo = Color(0,0,0); // Accumulator

// "Integrate over hemisphere" -> Sample many directions
for (Vector3 wi : SampleHemisphere()) {
    Color Li = TraceRay(p, wi); // Incoming light
    Color f  = Material.Eval(wo, wi); // BRDF
    float cosTheta = Dot(Normal, wi); // Geometry factor

    Lo += f * Li * cosTheta;
}
return Lo / NumSamples;
```

3 Color and Spectra

In standard web development, color is 'RGB(255, 0, 0)'. In physics, color is a continuous function of wavelength.

3.1 Spectral Power Distribution (SPD)

Light is a wave. An SPD is a graph showing how much energy exists at each wavelength (λ).

- **Laser**: A spike at one specific wavelength (e.g., 650nm).
- **Sunlight**: A smooth curve across all visible wavelengths.

3.2 XYZ and RGB

We cannot simulate infinite wavelengths. We need to compress this data. Humans have 3 types of cones in our eyes. This means we can compress the infinite SPD into just 3 numbers.

XYZ Color Space: The standard "human vision" coordinate system. **RGB Color Space**: A device-dependent system (your monitor).

The Conversion Pipeline:

1. **Simulation**: PBRT v4 does everything using **Spectra** (sampled at specific wavelengths), not RGB. This is more accurate for dispersion (prisms) and complex materials.
2. **Film**: At the very end, the spectral samples are converted to **XYZ**.
3. **Display**: XYZ is converted to **sRGB** (or Rec2020) to be displayed on your screen.

Why not just use RGB?

Multiplying RGB values is mathematically wrong for physics.

- Real physics: Multiply the SPD curves.
- RGB hack: Multiply R*R, G*G, B*B.

For simple matte paints, RGB is "close enough." For complex phenomena like skin, water, or iridescent beetles, RGB fails. PBRT v4 uses full spectral rendering to fix this.

Summary for the Developer

1. **Know your Units**:
   - Flux (Φ) = Power (Lightbulb wattage).
   - Irradiance (E) = Light arriving at surface.
   - Radiance (L) = Light traveling along a ray (Pixel color).
2. **The Integral is a Loop**: The "Rendering Equation" is just a loop summing up light * material * cosine.
3. **Spectra ¿ RGB**: Real light is a wave. RGB is just a compression format for human eyes. PBRT simulates the waves (Spectra) and only converts to RGB at the very end.